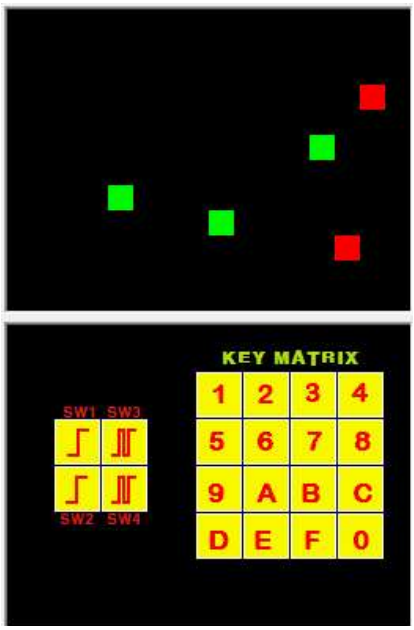
실험 7. Multi-Tasking Box with Mutex

1. 목적

이 실험에서는 지난 실험 결과 움직이는 6개의 Ball이 서로 겹쳐지면서 다음과 같이 Ball 6개가 화면에 항상 나오지는 않는 문제를 운영체제의 Mutex (Semaphore)를 이용하여 해결하는 실험이다.



이 실험에서는 지난 실험 결과를 이용하며, 가로, 세로로 움직이는 Ball이 만나는 9개의 지점에서 두 Ball이 겹쳐지는 지점에 한 Ball만 진입을 하고 그 Ball이 지나가면 다른 Ball이 지나가서 항상 6개의 Ball이 화면에 보이도록 만드는 상호배제를 구현하는 실험이다.

2. 실험 내용 설명

Ball을 화면에 그리는 방법은 지난 실험 내용을 참조하고. 이 실험에서는 상호배제를 위해 세마포에 관하여 설명한다.

세마포를 이용하기 위해서는 RTOS가 지원하는 Semaphore Primitive를 이용한다. http://www.freertos.org/의 <API Reference> - <Semaphore/Mutexes> 섹션을 참고하면 된다. 앞 홈페이지에는 각 API에 대한 설명과 예제가 나와 있다. (sourceforge 자료실의 한글 버전 참조)

주요 사용 API는 vSemaphoreCreateBinary(), xSemaphoreTake(), xSemaphoreGive() 이다. 이 함수들을 이용하려면 “semphr.h"를 #include 해야 한다.

(1) **vSemaphoreCreateBinary**(xSemaphoreHandle xSemaphore)

Macro that creates a semaphore by using the existing queue mechanism. The queue length is 1 as this is a binary semaphore. The data size is 0 as we don't want to actually store any data - we just want to know if the queue is empty or full.

A binary semaphore need not be given back once obtained, so task synchronization can be implemented by one task/interrupt continuously 'giving' the semaphore while another continuously 'takes' the semaphore. (즉 생성한 뒤, 최초의 Take가 성공한다)

A Binary semaphore is assigned to variables of type **xSemaphoreHandle** and can be used in any API function that takes a parameter of this type.

Parameters:
xSemaphore Handle to the created semaphore. Should be of type xSemaphoreHandle.

(2) **xSemaphoreTake** (xSemaphoreHandle xSemaphore,
portTickType xBlockTime
)

Macro to obtain a semaphore. The semaphore must have previously been created with a call to vSemaphoreCreateBinary().

Parameters:
xSemaphore A handle to the semaphore being taken - obtained when the semaphore was created.
xBlockTime The time in ticks to wait for the semaphore to become available. The macro portTICK_RATE_MS can be used to convert this to a real time. A block time of zero can be used to poll the semaphore. (우리 FreeRTOS의 경우 항상 0을 지정하여 NonBlocking 모드로 동작하도록 한다.)

Returns:
pdTRUE if the semaphore was obtained. **pdFALSE** if xBlockTime expired without the semaphore becoming available.

(3) **xSemaphoreGive** (xSemaphoreHandle xSemaphore)

Macro to release a semaphore. The semaphore must have previously been created with a call to vSemaphoreCreateBinary().

Parameters:
xSemaphore A handle to the semaphore being released. This is the handle returned when the semaphore was created.

Returns:
pdTRUE if the semaphore was released. **pdFALSE** if an error occurred. Semaphores are implemented using queues. An error can occur if there is no space on the queue to post a message - indicating that the semaphore was not first obtained correctly.

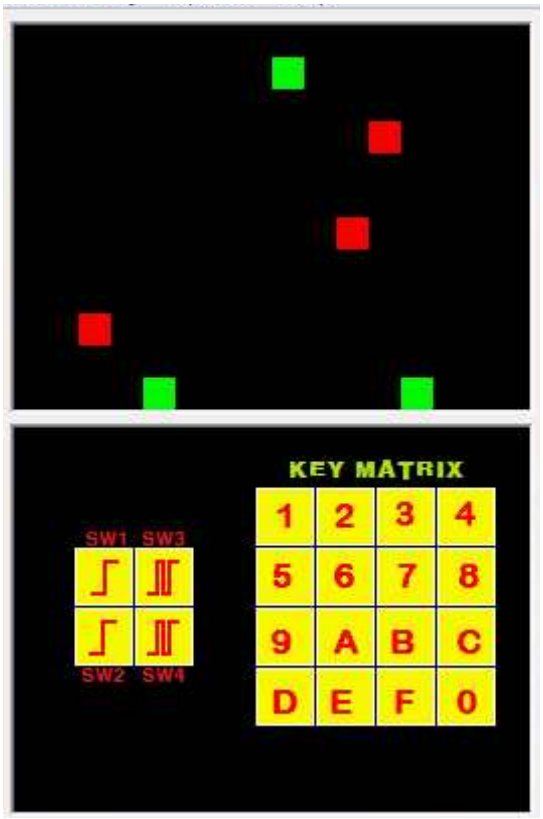
* 실제 vSemaphoreCreateBInary()는 매크로로 다음과 같이 정의된다.

```
#define vSemaphoreCreateBinary( xSemaphore ){\
    xSemaphore = xQueueCreate( ( unsigned portBASE_TYPE ) 1,
                               semSEMAPHORE_QUEUE_ITEM_LENGTH );\
    if( xSemaphore != NULL ){\
        xSemaphoreGive( xSemaphore );\
    }\
}
```

즉, Queue를 하나 할당 받고 (큐 길이는 1, semSEMAPHORE_QUEUE_ITEM_LENGTH - 큐 안의 데이터 길이는 0) xSemaphoreGive를 호출하여 세마포 하나를 바로 take 가능하도록 초기화 된다. 이후에 Queue에 데이터를 꺼내는 동작과 데이터를 넣는 동작으로 Semaphore 기능을 구현하고 있다.

3. 응용 실험 (과제)

과제는 지난 실험의 Ball 움직임에 세마포를 이용하여 'R'Key (오른쪽 뒤에 있는 스위치)가 눌려 있는 동안에만, Ball이 만나는 9개 지점에서 겹침없이 대기하고 지나가도록 만드는 것이다. (즉, R Key가 눌려있지 않다면 상호배제가 동작을 하지 않으므로 Ball이 겹쳐서 안보이는 상태이 가끔 발생하다가, R 키를 누르고 있으면 상호배제가 동작하여 모든 Ball이 화면에 보이고 겹치기 직전 대기를 하게 된다)



각 Ball를 그리는 Task는 다음과 같이 Delay 값을 정한다. (지난 실험의 Sample과 다른 방법으로 Task Parameter를 설정했다면. Ball을 그린 후, 아래 구조의 제일 오른쪽에 있는 시간(msec) 만큼 대기했다가 움직이도록 한다)

```
40 struct parameters Param[NUM_TASK] = {
41     { "1", DIRECTION_RIGHT, 3,  COLOR_RED,   350 },
42     { "2", DIRECTION_RIGHT, 6,  COLOR_RED,   300 },
43     { "3", DIRECTION_RIGHT, 9,  COLOR_RED,   320 },
44     { "4", DIRECTION_DOWN,  4,  COLOR_GREEN, 370 },
45     { "5", DIRECTION_DOWN,  8,  COLOR_GREEN, 400 },
46     { "6", DIRECTION_DOWN,  12, COLOR_GREEN, 330 }
47 };
```

과제 Template의 수정해야 할 부분은 '<----' 와 같은 comment로 처리되어 있음
(semphore 핸들 선언, 초기화, 부분운 소스 참조)

```
static portTASK_FUNCTION(Ball_Task, pvParameters)
{
   struct parameters *p = (struct parameters *)pvParameters;
   int x, y, prevX, prevY;

   // 여기에 각 Ball의 초기위치 (x,y) 설정<-------- 지난 실험 내용

   while(1) {

      // 각 Ball이 교차점에 있고, R Key가눌렸다면 Semaphore take <--------

      draw_my_box(prevX, prevY, COLOR_BLACK);// 예전위치삭제
      draw_my_box(x, y, p->color);// 새위치에그림
      vTaskDelay(MSEC2TICK(p->delay));// Delay

      // 각 Ball이 교차점에 있으면, Semaphore Give <--------

      prevX = x; prevY = y;

      // 다음 위치 계산 <----- 지난 실험 내용
    }
}
```

4-1. 소스 및 샘플 프로그램

A. Template 소스는 자료실의 Ball-Mutex.zip에 있음
B. 미리 만들어 놓은 Binary는 자료실의 Ball-Mutex.nds 임

4-2. 소스 및 샘플 프로그램 (Github 이용)

소스는 Github의 Ball 소스를 check-out하여 과제를 수행한다.

https://github.com/hl1itj/nds-ide/tree/microprocessor/lab/ball-mutex

에서, Checkout하여 과제를 수행하고 (main.c를 수정하여 구현한다..)

(이 문서, 미리 build한 Ball-Mutex.nds는 doc directory에 있음)

https://github.com/hl1itj/nds-ide/tree/microprocessor/학번/ball-mutex

에서 각자 개발을 진행한다.